

Rev 1.0

SW Module Estimates for NICI

SW module estimates for NICI

James A. Hinds
March 11, 2002
Ver 1.0
Issued By: MKIR

Rev 1.0

SW Module Estimates for NICI

Revision Control

1.0 Initial

Table of contents

SW MODULE ESTIMATES FOR NICI.....	1
REVISION CONTROL	2
TABLE OF CONTENTS	3
1 DESCRIPTION.....	5
1.1 PURPOSE.....	5
1.2 SCOPE.....	5
2 RELATED DOCUMENTS, GLOSSARY AND ACRONYMS.....	6
2.1 REFERENCES.....	6
2.2 ABBREVIATIONS AND ACRONYMS	6
3 PHYSICAL SYSTEM INTERFACES.....	7
3.1 RELATED INTERFACE CONTROL DRAWINGS	7
3.2 MECHANICAL INTERFACE	7
3.3 OPTICAL INTERFACE	7
3.4 ELECTRONIC INTERFACE.....	7
3.5 MASS/BALANCE	7
3.6 THERMAL INTERFACE.....	7
4 SOFTWARE/CONTROL FUNCTION METRICS BY MODULE	8
4.1 GENERIC IC CLASSES AND NAMESPACES	8
4.1.1 <i>Logwindow</i>	8
4.1.2 <i>LOG</i>	8
4.1.3 <i>IC</i>	8
4.1.4 <i>Newinterp</i>	9
4.1.5 <i>Hail</i>	9
4.1.6 <i>CO – Command Object (a template class)</i>	9
4.1.7 <i>Watchdog</i>	9
4.1.8 <i>TIP</i>	10
4.1.9 <i>Summary for Section 4.1</i>	10
4.2 SPECIFIC COMMAND OBJECTS IN IC:	10
4.2.1 <i>AO – Based on CO</i>	10
4.2.2 <i>DataBase – Template Class – Based on CO</i>	10
4.2.3 <i>Lakeshore – Based on CO</i>	11
4.2.4 <i>Smartmotor – Template Class – Based on CO</i>	11
4.2.5 <i>LD – Linear Discrete Mechanism – Template Class – Based on Smartmotor</i>	11
4.2.6 <i>RC – Rotary Continuous Mechanism – Template Class – Based on Smartmotor</i>	11
4.2.7 <i>RD – Rotary Discrete Mechanism – Template Class – Based on Smartmotor</i>	12
4.2.8 <i>TTSM – Voice Coil Technology</i>	12
4.2.9 <i>FOCS – Based On LD</i>	12
4.2.10 <i>NDFW – Based on CO</i>	12
4.2.11 <i>FPMW – Based On RC</i>	12
4.2.12 <i>SMR – Based On RC</i>	12
4.2.13 <i>PMW – Based On RD</i>	12
4.2.14 <i>DW – Based On RD</i>	13
4.2.15 <i>C1FW – Based On RD</i>	13
4.2.16 <i>C2FW – Based On RD</i>	13
4.2.17 <i>PI – Based On RD</i>	13
4.2.18 <i>DOFF</i>	13
4.2.19 <i>IDAC</i> :.....	13
4.2.20 <i>Acquire</i>	13
4.2.21 <i>Summary for Section 4.2</i>	14

4.3	SPECIFIC DEVICE OBJECTS IN IC	14
4.3.1	<i>Image</i>	14
4.3.2	<i>Out (destination) handlers – 4 objects FF, FS, DHS, QL</i>	14
4.3.3	<i>SM (status monitor)</i>	15
4.3.4	<i>CLOCK</i>	15
4.3.5	<i>Summary for Section 4.3</i>	15
4.4	SPECIFIC DEVICE OBJECTS IN PIXEL SERVER	15
4.4.1	<i>Image</i>	15
4.4.2	<i>Out (destination) handlers – 4 objects FF, FS, DHS, QL</i>	16
4.4.3	<i>SM (status monitor)</i>	17
4.4.4	<i>Summary for Section 4.4</i>	17
4.5	GENERIC CLASSES AND SUPPORT IN PIXEL SERVER.....	17
4.5.1	<i>Summary for Section 4.5</i>	17
4.6	CODE SIZE SUMMARY FOR SECTIONS 4.1, 4.2, 4.3, 4.4 AND 4.5.....	17
4.7	CONVERSION OF SUMMARY TO HOURS OF PROGRAMMING.....	18
5	BEHAVIOUR	19
6	SCENARIOS	20
7	DETAILED DESCRIPTION	21
7.1	COMMAND DESCRIPTION	21
7.2	STATUS INFORMATION	21
7.3	ALARM CONDITIONS	21
7.4	DEBUGGING AND MAINTENANCE.....	21
7.5	SIMULATION	21
8	SAFETY ISSUES	23

1 Description

1.1 Purpose

Modern projects often contain a software development component. This component can become unmanageable for any number of reasons. The concept of software engineering is still a developing field. There are few dependable tools for analysis, modeling and creation. The concept of software creation as a craft is still a valid formula. Even so, some gauge of the amount of actual work to be done is important for tracking the development effort. To that end, these metrics have been useful to many managers: the estimate of the number of modules, their variables and procedures, as well as a rough estimate of the size of each. These metrics can be used to track the actual completion of portions of the software effort and to insure that the amount of work is matched by the time available to the programming team.

The software modules for the NICI instrument are described in various documents. This document collects the names, variables and gives an estimate for the expected size of each method (procedure) in a given module.

In addition an estimate of the “craft” required. While the goal of software development from a management perspective is to reduce it to a engineering effort, with solid metrics and no uncertainties. Realistically, however, the software is special in that unknowns from the rest of the system usually reflect back on the software. These unknowns are among the following: new understanding of hardware capabilities (or the lack thereof), new realizations of cost saving alternatives in purchasing, inability of third-party suppliers to deliver thus causing re-evaluation of alternate hardware. There are always “negative features” of support libraries and operating systems that surface late in the development cycle. And finally, the effective speed of the software may dictate rework: from “tuning” to complete re-write.

This leads to an uncertainty which needs to be addressed in these estimates. Each section will end with an estimate of the re-work potential for the lines of code. There are 3 classes: boilerplate, journeyman, and craftsman.

- Boilerplate is considered to be the code that must be present for syntactic, semantic or documentation consistency. For example, comments that trigger automatic generation of html documentation, or “glue” routines. This code is very stable, however commentary needs to accurately reflect changes in journeyman or craftsman code.
- Journeyman code is that portion of the system that is fairly routine, and usually does not impact performance or reliability. For example, the code that fills in the FITS headers of data sets is straightforward: The kinds of statements are limited and easily verified. The correctness, once in place, is above suspicion.
- Craftsman code is that portion that may have subtle side effects, or must be written in some non-obvious way to perform correctly. This code usually heavily impacts the reliability or throughput of the system. Often this code is the focus of more development time than either of the other two categories. This code may need to be re-worked several times.

Both boilerplate and journeyman code are easily managed. The amount of management attention to these areas gives good results as far as progress to the goal. In a “good” software design the amount of code needing the craftsman touch should be a small percentage of the total code size. Part of each section consists of estimates of the amount of boilerplate, journeyman and craftsman code required.

1.2 Scope

This document describes only the modules in the instrument controller and pixel servers for the NICI instrument. It does not cover existing system software or libraries that are used by the development team. In particular, the following software items are not specified here:

- Tcl/Tk (scripting environment for graphic and non-graphic distributed applications)
- Linux (or other OS)
- Libraries for Socket (TCP/IP) communication.
- Libraries for memory management.
- Libraries for I/O formatting (e.g. Cfitsio), I/O handling (including DHS libraries) or scientific computation.
- Libraries supplied by external sources (e.g. Device drivers)
- Supplied subassemblies (e.g. Adaptive Optics processor)

2 Related Documents, glossary and acronyms

2.1 References

- [1] SW Control of temperature system for redstar III controller. MKIR document, Author Jim Hinds.
- [2] NICI mechanism control. MKIR document, Author Jim Hinds
- [3] NICI2AO. MKIR document, Author Jim Hinds
- [4] NICI2DHS. MKIR document, Author Jim Hinds
- [5] Software Array Control – lite. MKIR document, Author Jim Hinds
- [6] NICI sw architecture. MKIR document, Author Jim Hinds

2.2 Abbreviations and Acronyms

AO Adaptive Optic subsystem.

DHS Data Handling System

EPICS Experimental Physics and Industrial Control System

IOC Input-Output Controller

N/A Not applicable here

NICI Near Infrared Coronagraphic Imager

QLT Quick Look Tool, part of the DHS

RISC

SBC Single Board Computer

TBD To Be Determined

TCL/TK universal scripting system (this will include the object oriented extension Itcl)

TCS Telescope Control System

T/T Tip-Tilt

3 Physical System Interfaces

3.1 *Related Interface Control Drawings*

Not applicable

3.2 *Mechanical Interface*

Not applicable

3.3 *Optical Interface*

Not applicable

3.4 *Electronic Interface*

Not applicable

3.5 *Mass/Balance*

Not applicable

3.6 *Thermal Interface*

Not applicable

4 Software/Control Function Metrics by Module

Each module description will have 2 or 3 sections. The first section is the estimate of the number of variables local to that object, if the names have already been selected, then they are listed. The second section is the estimate of the methods or local procedures. The externally visible names have already been documented in the referenced papers. Each method is shown by a line as the following:

Convert – 15 lines TCL – 15 lines of C

This indicates that the method is to be implemented as a 15 line (including comments for documentation engines) TCL routine. Also this routine will invoke the support of some C language code that is estimated at 15 lines of code. The interface to these supporting routines is automatically generated by a program called SWIG, and is therefore not specified in the lines of code to be written.

The third and optional section specifies any low level libraries that are needed to perform the functions of the object. For example the library to access the DHS system would be mentioned in this section. The interface to these supporting routines is automatically generated by a program called SWIG, and is therefore not specified in the lines of code to be written.

4.1 Generic IC Classes and Namespaces

These following are to open input channels from users, get the input messages, route the messages to the appropriate Command Objects, and provide the return channel for results or error messages to the user.

These generic classes are written and tested, for this reason the estimates of code size is omitted. The code may be viewed on-line at http://64.65.110.34/mkir_controller/ and <http://64.65.110.34/cgi-bin/cvsweb.cgi/socon/server/ic/server.tcl>

The CO class works with the TCL environment to process all 'configure' and 'cget' commands for all Command Objects. There is no separate 'configure' or 'cget' method for any of the command objects.

These generic classes contain 35% boilerplate, 50% journeyman code and 15% craftsman quality code. The boilerplate and journeyman portions of this code will not likely need changes. The potential for re-work of the craftsman code is on the order of 100 lines of TCL. The total for the Boilerplate is 233 lines, Journeyman 333 lines.

4.1.1 Logwindow

Variables 4:

- Mywindow
- Mytext
- Session
- Mysession

Methods 7:

- Destructor
- Constructor
- Mesg
- Warn
- Err
- Dbg
- Inmsg

4.1.2 LOG

Variables 2:

- Logfile
- Needtimestamp

Methods

- Watchdog
- mesg

4.1.3 IC

Variables 1:

- All_co
- Methods 2:**
- Set_error
 - Add_co

4.1.4 Newinterp

- Variables 3:**
- Interpreter
 - Channel
 - Log
- Methods 4:**
- Replychannel
 - Destructor
 - Constructor
 - Evaluate

4.1.5 Hail

- Variables 4:**
- Hailing_channel
 - Interpreter
 - Activecommand
 - All_interps
- Methods:**
- Broadcast
 - Replychannel
 - Constructor
 - Destructor
 - Newhail
 - Process_message

4.1.6 CO – Command Object (a template class)

- Variables 5:**
- testing
 - replychannel
 - printname
 - state
 - paused
- Methods 17:**
- Constructor
 - Itest
 - Testcheck
 - Replychannel
 - Read_only
 - Checked_var
 - Accept
 - Reject
 - Test
 - Pause
 - Resume
 - Abort
 - Pause_or_abort
 - Enteridle
 - Entererror
 - Enteractive
 - recover

4.1.7 Watchdog

Class overhead (boilerplate) 20 lines.

- Variables 2:**

Rev 1.0

SW Module Estimates for NIC1

- AllMotors

Methods

- Scan – 25 lines TCL

Total 49 lines. Boilerplate code 70% (34 lines) , Journeyman code 30% (15 lines), Craftsman 0% (0 lines)

4.1.8 TIP

Class overhead (boilerplate) 20 lines.

Variables 2:

- TipID
- ControlPort

Methods

- Constructor – 25 lines TCL
- Send – 5 lines TCL
- Recover – 100 lines TCL

A port of code to service the TIP from the Navy Astrocams project. Requires considerable craftsman style re-work if new manufacturer/model is used.

Total 159 lines. Boilerplate code 70% (111 lines) , Journeyman code 30% (48 lines), Craftsman 0% (0 lines)

4.1.9 Summary for Section 4.1

Section	Total	checksum	Boilerplate	Journeyman	Craftsman
4.1.1 to 4.1.6	666	102	233	333	100
4.1.7	49	120	34	15	0
4.1.8	159	42	111	48	0
	874	874	378	396	100

Total lines = 874, Boilerplate 378, journeyman 396, craftsman 100.

4.2 Specific Command Objects in IC:

These classes are dedicated to specific device functionality, These have not been written, so an estimate of the code size is included.

4.2.1 AO – Based on CO

Class overhead (boilerplate) 20 lines.

Variables 4:

- DetectorCounts (vector of counts)
- Loopgain
- Tiltstate
- Focuslimit
- Stroke
- tiltlimit

Methods

- Constructor – 20 lines TCL
- SendtoAO – 5 lines TCL
- ReceiveFromAO – 25 lines TCL
- Poll – 25 lines TCL
- Pause – 10 lines TCL
- Focus – 10 lines TCL
- Tilt – 10 lines TCL
- Strokeset – 25 lines TCL

Total 83 lines. Boilerplate code 50% (42 lines) , Journeyman code 35% (30 lines), Craftsman 15% (11 lines)

4.2.2 DataBase – Template Class – Based on CO

Class overhead (boilerplate) 30 lines

Variables 1:

- Dictionary

Methods 2

- constructor – 15 lines TCL
- destructor – 10 lines TCL

Rev 1.0

SW Module Estimates for NICI

Note: Command Objects instantiated from this class will only hold data values that may be important for other objects of the system. It will be instantiated as the Command Object "Telescope" to hold azimuth, altitude, and rotator values. It will also be instantiated as an object to hold header data (FITS or DHS)

Total 56 lines. Boilerplate code 50% (28 lines) , Journeyman code 50% (28 lines), Craftsman 0% (0 lines)

4.2.3 Lakeshore – Based on CO

Class overhead (boilerplate) 20 lines.

Variables 4:

- S
- R
- Nsensors
- TIP

Methods

- Constructor – 20 lines TCL
- Send – 5 lines TCL
- ReceiveFromTip – 5 lines TCL
- Poll – 25 lines TCL

Total 83 lines. Boilerplate code 50% (42 lines) , Journeyman code 35% (30 lines), Craftsman 15% (11 lines)

4.2.4 Smartmotor – Template Class – Based on CO

Class overhead (boilerplate) 20 lines.

Variables 2:

- FirmwarePath
- TIP

Methods

- Constructor – 20 lines TCL
- Send – 5 lines TCL
- ReceiveFromTip – 5 lines TCL
- StillMoving – 5 lines TCL

Total 57 lines. Boilerplate code 70% (40 lines) , Journeyman code 25% (14 lines), Craftsman 5% (3 lines)

4.2.5 LD – Linear Discrete Mechanism – Template Class – Based on Smartmotor

Linear Discrete Mechanisms present little new code. Most is ported from the Astrocarn project. The old code must be adapted to the object model by using proper boilerplate.

Class overhead (boilerplate) 25 lines.

Variables 10:

- NPositions
- PositionNames
- PortAddress
- Var internal values.

Methods

- Park – 15 lines TCL (new)
- Move – 15 lines TCL (new)
- Existing code size approx. 100 lines TCL

Note: Uses Pupil imager smartmotor code from Astrocarn

Total 75 lines. Boilerplate code 70% (53 lines) , Journeyman code 30% (22 lines), Craftsman 0% (0 lines)

4.2.6 RC – Rotary Continuous Mechanism – Template Class – Based on Smartmotor

Rotary Continuous Mechanisms present some new code. Most is ported from the Astrocarn project. The old code must be adapted to the object model by using proper boilerplate.

Class overhead (boilerplate) 25 lines.

Variables 10:

- Resolution
- LashSlack
- PortAddress

Var internal values.

Methods

- Move – 100 lines TCL

Rev 1.0 SW Module Estimates for NICI

- Park – 20 lines TCL

Note: Uses Pupil imager smartmotor code from Astrocam.

Total 165 lines. Boilerplate code 60% (99 lines) , Journeyman code 30% (50 lines), Craftsman 10% (16 lines)

4.2.7 RD – Rotary Discrete Mechanism – Template Class – Based on Smartmotor

Linear Discrete Mechanisms present little new code. Most is ported from the Astrocam project. The old code must be adapted to the object model by using proper boilerplate.

Class overhead (boilerplate) 25 lines.

Variables 10:

- NPositions
- PositionNames
- PortAddress
- Var internal values.

Methods

- Park – 15 lines TCL (new lines only)
- Move – 15 lines TCL (new lines only)
- Existing code size approx. 100 lines TCL

Note: Also uses filter wheel smartmotor code from Astrocam.

Total 75 lines. Boilerplate code 70% (53 lines) , Journeyman code 30% (22 lines), Craftsman 0% (0 lines)

4.2.8 TTSM – Voice Coil Technology

Class overhead (boilerplate) 25 lines.

Variables 5:

- PortAddress
- Var internal values.

Methods

- Park – 10 lines TCL
- Move – 25 lines TCL
- ReceiveFromVC – 10 lines TCL
- StillMoving – 5 lines TCL

Simple communication with smart TCP/IP based DAC voice coil mechanism

Total 85 lines. Boilerplate code 30% (26 lines) , Journeyman code 40% (33 lines), Craftsman 30% (26 lines)

4.2.9 FOCS – Based On LD

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.10 NDFW – Based on CO

Class overhead (boilerplate) 20 lines.

Variables 1:

- Position

Methods

- Move – 15 lines TCL
- StillMoving – 15 lines TCL
- Park – 5 lines TCL

Simple communication with blind waits for completion

Total 57 lines. Boilerplate code 30% (18 lines) , Journeyman code 20% (11 lines), Craftsman 50% (28 lines)

4.2.11 FPMW – Based On RC

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.12 SMR – Based On RC

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.13 PMW – Based On RD

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.14 DW – Based On RD

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.15 C1FW – Based On RD

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.16 C2FW – Based On RD

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.17 PI – Based On RD

Class overhead (boilerplate) 30 lines.

No new Variables or Methods

Total 30 lines. Boilerplate code 100% (30 lines) , Journeyman code 0% (0 lines), Craftsman 0% (0 lines)

4.2.18 DOFF

Class overhead (boilerplate) 25 lines.

Variables 1:

- position

Methods

- Null – 5 lines TCL
- Compensation – 20 lines TCL
- Move – 10 lines TCL
- Moveto – 10 lines TCL

Total 72 lines. Boilerplate code 30% (22 lines) , Journeyman code 0% (0 lines), Craftsman 70% (50 lines)

4.2.19 IDAC:

Class overhead (boilerplate) 25 lines.

Variables 1:

- Dacbias

Methods 2:

- Sendbias - 15 lines TCL
- Readback - 20 lines TCL

Total 60 lines. Boilerplate code 50% (30 lines) , Journeyman code 45% (27 lines), Craftsman 5% (3 lines)

4.2.20 Acquire

Class overhead (boilerplate) 25 lines.

Variables 10:

- Subarray (an vector of image rectangle specs)
- Mode (keyword)
- ltime (integration time)
- Rtime (reset time)
- Ndr (non-destructive readout)
- Exposures
- Sequence (DHS ID)

Methods 7

- Acquire – 50 lines TCL
- Stop – 50 lines TCL
- Abort – 50 lines TCL
- Create_decode_map – 500 lines C code (can be extracted from 'Astrocam')
- Create_clocking_pattern – 700 lines C code (core can be extracted from 'Astrocam')
- Mode_arc_d – 100 lines TCL
- Mode_arc_s – 80 lines TCL

Total lines 1565. Boilerplate code 30% (470 lines) , Journeyman code 40% (626 lines), craftsman code 30% (469 lines), Mode_arc_d and Mode_arc_s have a higher percentage of craftsman quality code.

4.2.21 Summary for Section 4.2

Section	Total	checksum	Boilerplate	Journeyman	Craftsman
4.2.1	83	83	42	30	11
4.2.2	56	56	28	28	0
4.2.3	83	83	42	30	11
4.2.4	57	57	40	14	3
4.2.5	75	75	53	22	0
4.2.6	165	165	99	50	16
4.2.7	75	75	53	22	0
4.2.8	85	85	26	33	26
4.2.9	30	30	30	0	0
4.2.10	57	57	18	11	28
4.2.11	30	30	30	0	0
4.2.12	30	30	30	0	0
4.2.13	30	30	30	0	0
4.2.14	30	30	30	0	0
4.2.15	30	30	30	0	0
4.2.16	30	30	30	0	0
4.2.17	30	30	30	0	0
4.2.18	72	72	22	0	50
4.2.19	60	60	30	27	3
4.2.20	1565	1565	470	626	469
	2673	2673	1163	893	617

Total lines = 2673, boilerplate 1163, journeyman 883, craftsman 617

4.3 Specific device objects in IC

These low-level objects are not user accessible, therefor are not Command Objects, but are used directly by the corresponding user interface CO.

4.3.1 Image

Class overhead 20 lines.

Variables 1:

- Pixels

Methods 12:

- Communicate_with_PS – forwards command to Pixel server via TCP/IP – 20 lines TCL
- Receive_report_from_ps – callback from Pixel Server – 10 lines TCL
- New – 5 lines TCL
- Flush – 5 lines TCL
- Set – 5 lines TCL
- Addto – 5 lines TCL
- Add – 5 lines TCL
- Subtract – 5 lines TCL
- Pedestal – 5 lines TCL
- Multiplyby – 5 lines TCL
- Inverse – 5 lines TCL
- Sendto – 5 lines TCL
- Report – 5 lines TCL

Total 102 lines, Boilerplate 50% (51 lines), journeyman 45% (45 lines), craftsman 5% (6 lines).

4.3.2 Out (destination) handlers – 4 objects FF, FS, DHS, QL

Class overhead 25 lines.

Variables 2:

Rev 1.0

SW Module Estimates for NICI

- Destination – array of handlers and destination Ids (e.g. \$destination(FF) = “/some/file/name.fts”)

Methods 14

- Communicate_with_ps – forwards command to PS via TCP/IP – 20 lines TCL
- Send_map_to_ps – forwards decode map when changed by configure 20 lines TCL
- Receive_report_from_ps – callback from Pixel Server – 10 lines TCL
- Open – 5 lines TCL
- Close – 5 lines TCL
- New – 5 lines TCL
- Convert – 5 lines TCL
- Decode – 5 lines TCL
- Integer – 5 lines TCL
- String – 5 lines TCL
- Float – 5 lines TCL
- Out – 5 lines TCL
- Map – 5 lines TCL
- Report – 5 lines TCL

Total 120 lines, Boilerplate 50%, journeyman 45%, craftsman 5%, 60, 54 and 6 lines respectively.

4.3.3 SM (status monitor)

Class overhead 20 lines.

Methods 1:

- Receive_report_from_ps – callback from Pixel Server – 20 lines TCL

Total 42 lines, Boilerplate 50%, journeyman 45%, craftsman 5%, 21, 18 and 3 lines respectively.

4.3.4 CLOCK

Class overhead 20 lines.

Variables 10: (est)

Methods 6:

- Clocktalk – pass command to TCP/IP clocker – 15 lines TCL
- Sync – 5 lines TCL
- DSCP – 5 lines TCL
- SSCM – 5 lines TCL
- RESET – 5 lines TCL
- SCAN – 5 lines TCL

Total 80 lines, Boilerplate 50%, journeyman 45%, craftsman 5%, 40, 36 and 4 lines respectively.

4.3.5 Summary for Section 4.3

Section	Total	checksum	Boilerplate	Journeyman	Craftsman
4.3.1	102	102	51	45	6
4.3.2	120	120	60	54	6
4.3.3	42	42	21	18	3
4.3.4	80	80	40	36	4
	344	344	172	153	19

Total lines = 344, boilerplate 172, journeyman 153, craftsman 19

4.4 Specific device objects in Pixel Server

These low-level objects correspond to the client objects in the IC. These modules are identical in each of the two Pixel Servers of NICI.

4.4.1 Image

Class overhead 20 lines.

Variables 1:

- Pixels

Methods 12:

- Communicate_with_IC – forwards reply to IC client via TCP/IP – 20 lines TCL

Rev 1.0

SW Module Estimates for NICI

- Receive_report_from_IC – callback from IC – 10 lines TCL
- New – 25 lines TCL – 100 lines C
- Flush – 15 lines TCL – 25 lines C
- Set – 10 lines TCL – 10 lines C
- Addto – 10 lines TCL – 15 lines C
- Add – 10 lines TCL – 15 lines C
- Subtract – 10 lines TCL – 15 lines C
- Pedestal – 10 lines TCL – 15 lines C
- Multiplyby – 10 lines TCL – 15 lines C
- Inverse – 10 lines TCL – 15 lines C
- Sendto – 10 lines TCL – 15 lines C
- Report – 10 lines TCL

Total 397 lines, Boilerplate 65%, journeyman 25%, craftsman 10%, 257, 100 and 40 lines respectively.

4.4.2 Out (destination) handlers – 4 objects FF, FS, DHS, QL

FF and FS are similar and will share code. The socket and file specification will be passed to a single back end as parameters. Likewise with QL and DHS.

4.4.2.1 QL & DHS

Class overhead 25 lines.

Variables 2:

- Destination – array of handlers and destination Ids (e.g. \$destination(FF) = “/some/file/name.fts”)

Methods 14

- Communicate_with_ps – forwards command to PS via TCP/IP – 20 lines TCL
- Send_map_to_ps – forwards decode map when changed by configure
- Receive_report_from_ps – callback from Pixel Server – 10 lines TCL
- Open – 15 lines TCL
- Close – 5 lines TCL
- New – 15 lines TCL
- Convert – 15 lines TCL – 15 lines of C
- Decode – 5 lines TCL – 25 lines of C
- Integer – 5 lines TCL – 15 lines of C
- String – 5 lines TCL – 15 lines of C
- Float – 5 lines TCL – 15 lines of C
- Out – 15 lines TCL
- Map – 5 lines TCL
- Report – 5 lines TCL

Supporting libraries: DHS interface library via SWIG interface

Total 257 lines, Boilerplate 65%, journeyman 25%, craftsman 10%, 167, 73 and 17 lines respectively.

4.4.2.2 FS & FF

Class overhead 25 lines.

Variables 2:

- Destination – array of handlers and destination Ids (e.g. \$destination(FF) = “/some/file/name.fts”)

Methods

- Communicate_with_ps – forwards command to PS via TCP/IP – 20 lines TCL
- Send_map_to_ps – forwards decode map when changed by configure
- Receive_report_from_ps – callback from Pixel Server – 10 lines TCL
- Open – 15 lines TCL
- Close – 5 lines TCL
- New – 15 lines TCL
- Convert – 15 lines TCL – 15 lines of C
- Decode – 5 lines TCL – 25 lines of C
- Integer – 5 lines TCL – 15 lines of C
- String – 5 lines TCL – 15 lines of C
- Float – 5 lines TCL – 15 lines of C

Rev 1.0

SW Module Estimates for NIC1

- Out – 25 lines TCL
- Map – 5 lines TCL
- Report – 5 lines TCL

Supporting libraries: CFITSIO interface library via SWIG interface

Total 257 lines, Boilerplate 65%, journeyman 25%, craftsman 10%, 167, 73 and 17 lines respectively.

4.4.3 SM (status monitor)

Class overhead 20 lines.

Methods 2:

- Poll – 25 lines TCL – 25 lines C
- Mark_buffer_in_use – 10 lines TCL
- Mark_buffer_free – 10 lines TCL
- Send report to IC – 20 lines TCL

Total 110 lines, Boilerplate 65%, journeyman 25%, craftsman 10%, 71, 27 and 12 lines respectively.

4.4.4 Summary for Section 4.4

Section	Total	checksum	Boilerplate	Journeyman	Craftsman
4.4.1	257	257	167	73	17
4.4.2	257	257	167	73	17
4.4.3	110	110	71	27	12
	624	624	405	173	46

Total lines = 624, Boilerplate 405, journeyman 173, craftsman 46.

4.5 Generic classes and support in Pixel Server

These classes will be cloned from the IC's generic classes. The Pixel Servers tasks of managing the I/O, dispatching the commands and providing development/debug framework are identical to the needs of the IC. While the IC's classes perform more services and routing than is required, they are certainly sufficient for the task. And they are already written, a big bonus.

The classes that are expected to be used here are:

- Logwindow
- LOG
- IC (renamed PS)
- Newinterp
- Hail
- CO

These code required for those classes will not be enumerated here, but only summarized.

4.5.1 Summary for Section 4.5

Total lines = 666, Boilerplate 333, journeyman 200, craftsman 133.

4.6 Code Size Summary for Sections 4.1, 4.2, 4.3, 4.4 and 4.5

4.1 – Total lines = 874, Boilerplate 378, journeyman 396, craftsman 100.

4.2 – Total lines = 2673, boilerplate 1163, journeyman 883, craftsman 617

4.3 – Total lines = 344, boilerplate 172, journeyman 153, craftsman 19

4.4 – Total lines = 624, Boilerplate 405, journeyman 173, craftsman 46.

4.5 – Total lines = 666, Boilerplate 333, journeyman 200, craftsman 133.

4.1	874	378	396	100
4.2	2673	1163	883	617
4.3	344	172	153	19
4.4	624	405	173	46
4.5	666	333	200	133
totals	5181	2451	1805	915

Total lines = 5181, boilerplate 2451, journeyman 1805, craftsman 915

4.7 Conversion of Summary to Hours of Programming

We currently estimate that 50 lines of boilerplate code takes 8 hours (6.25 lines / hour). Journeyman level code takes three times longer or 15 lines per 8 hours (1.875 lines /hour). Finally, craftsman level code takes three times longer at 5 lines per 8 hours (0.625 lines / hour).

With this in mind the total programming for the IS and IC (as well as smartmotors, TIP, and temperature monitor) would be:

Boilerplate: $2451 \text{ lines} / 6.25 = 392 \text{ hours}$

Journeyman: $1805 / 1.875 = 963 \text{ hours}$

Craftsman: $915 / 0.625 = 1464 \text{ hours}$

This yields a total of 2818 hours or 352 man-days.

As with all estimates, there are no fixed metrics. Each portion of this analysis has some unstated uncertainty. At any step there may be a 10 to 20 percent tolerance factor. By statistical reasoning, the final numbers could vary from these estimates by 50 to 100 man-days.

5 Behaviour

N/A

6 Scenarios

N/A

7 Detailed description

N/A

7.1 Command description

N/A

7.2 Status information

N/A

7.3 Alarm conditions

N/A

7.4 Debugging and Maintenance

N/A

7.5 Simulation

Each physical device or subsystem may need to be simulated during the software development process. Unfortunately, the value of such simulation code depends mainly on how closely it mimics the actual operation of the real device.

Simulated system	Entry Point	Code size for simulation
Animatics Smart Motors	Rotate to home	10 lines TCL
	Rotate to Detent	10 lines TCL
	Rotate amount	10 lines TCL
	Version	5 lines TCL
	Go fwd to home	10 lines TCL
	Report at home	10 lines TCL
AO system	each command	50 Lines TCL + 5 line TCL stub for each command
Array Clocker	scan	5 line TCL Stub
Fiber Optic Receiver	Generate_image	50 lines of C code to generate dummy image data
DHS system	dhsInit	Each stub estimated at 5 lines of C code
	dhsCallbackSet	
	dhsEventLoop	
	dhsDebugLevel	
	dhsConnect	
	dhsBdCtl	
	dhsDsNew	
	dhsBdAttributeAdd	
	dhsBdPut	
	dhsWait	
	dhsBdDsFree	

Rev 1.0

SW Module Estimates for NICI

	dhsBdCtl	
	dhsBdFrameNew	

8 Safety Issues

At present no safety issues have been identified relating to this interface.